Embedded Platform Comparison

Kylee Burgess, Arizona State University

Kylee Burgess is an accelerated master's student graduating in May of 2019 with a focus in electrical systems engineering and software engineering. She also has a BSE with a primary focus in electrical systems engineering and a secondary focus in software engineering from Arizona State University, Ira A. Fulton Schools of Engineering at the Polytechnic campus. Her research interests include electrical systems engineering, software engineering, education, and learning tools. She will be moving to Sunnyvale, CA to start a job as an Electrical Engineering Intern at 219 Design, a small engineering consulting company, in June of 2019.

Table of Contents

Introduction

There are many development boards for electrical engineering projects available on the market. “[C]hoosing development kits [is] a super-complex decision, with lots of technical attributes that don’t seem to make a lot of sense, especially if you’re just getting started with electronics” (Teel, 2018). This paper will assume the development board is being used for the full project, without switching to the target board.

Cost, features, programming language, and development tools are a few of the many important comparison points to consider. Cost of development boards is an important factor to consider in the overall cost of the project, potentially taking necessary funds from other components. The features included on the development board are important because without certain pins or communication options additional equipment may be required − increasing cost and development time. For example, in order to read in an analog input value with a development board that only contains digital I/O pins, an external analog to digital converter (ADC) will be required. The programming language and integrated development environment (IDE) are important due to familiarity, ease of learning, and ease of use. Familiarity with an existing development environment can reduce or eliminate the time commitment to learn a new one, speeding up overall development time for the project. However, when choosing an unfamiliar programming language and IDE, the availability of documentation is very important. Documentation or community assistance can help ensure the project does not come to a standstill due to complications.

For this project, five different platforms were compared across the aforementioned four areas. The development boards used were the *TinyFPGA Bx*, *PSoC 5LP*, *Arduino Nano*, *Trynkit Husky*, and the *NodeMCU v2*. This paper includes details as to what these platforms include, why they were chosen, what worked and did not work, complications that arose, and conclusions for each platform. Then, a comparison is drawn between the development boards based on their cost, features, programming language, and IDE.

Motivation
During my time applying for post-graduation jobs, many employers were drawn to the FPGA/Verilog and Embedded C written on my resume. However, although I had taken classes on these subjects and/or been introduced to them through an internship, I found myself unable to answer many of their interview questions. Reading papers and related material found online also proved ineffective. “While a lot of guides offer extensive technical guidance in choosing between different variants (Arduino, Raspberry Pi, BeagleBone), there aren’t many guides that explain the decisive factors that you need to consider while choosing a development board” (Teel, 2018). Before graduating, I wanted to gain hands-on experience using FPGA and Embedded C platforms. This paper aims to provide personal experience comparisons of five different development platforms in order to provide readers with a comprehensive guide to determine which platform fits their projects’ needs.

Methods
A simple robotics project was created to compare development on the different platforms. The project included digital input (a button), digital output (a solenoid), analog input (a potentiometer), analog output (a servo), and a display (an LCD screen). This project was chosen because it covered a variety of aspects using a complete physical project. To complete the project, the following steps were completed for each platform:

- Each development board started by installing or setting up the IDE.
- Digital IO was tested using a button and an LED.
- Analog in and out was implemented with a potentiometer and a servo.
- The LCD screen was tested.
- All tested components were integrated with the solenoid on the final robot.

The differences and findings for the development boards will be provided in the Results section.

Figure 1 below is a schematic for the robotics project. Labeled nets within these figures connect to different pins depending on the development board being used. The pins used are outlined in Table 1.

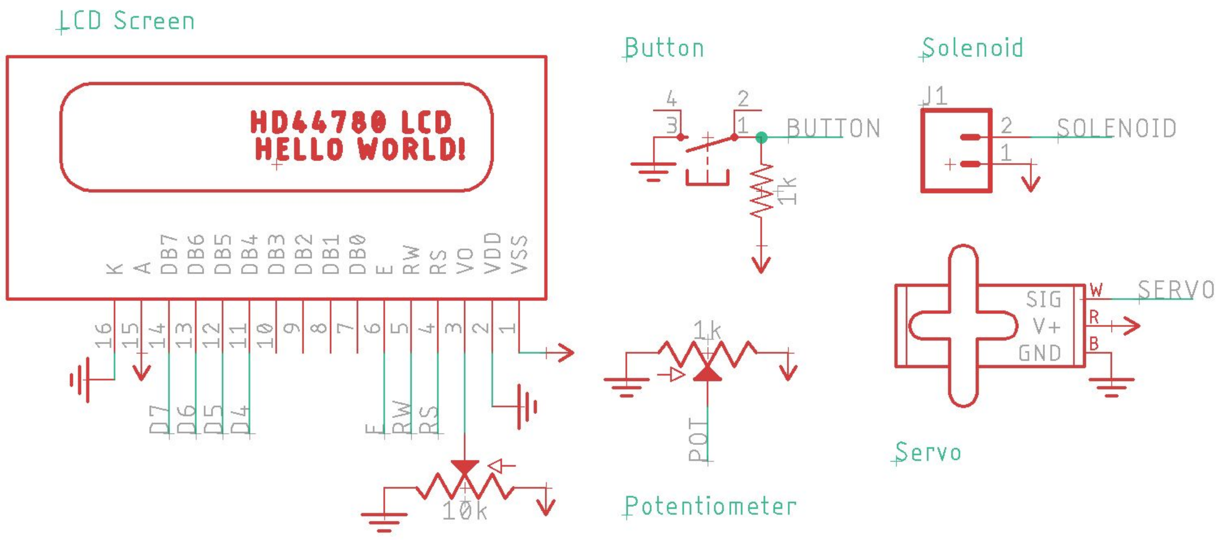


Figure 1: Schematic

Table 1: Net Connections for each Development Board

Pins	TinyFPGA	Arduino Nano	PSoC 5LP	Trynkit Husky	NodeMCU
RS	--	D7	2.5	PD0	TX - GPIO1
RW	--	GND	2.6	GND	GND
E	--	D6	2.4	PD1	RX - GPIO3
D4	--	D5	2.0	PD4	D6 - GPIO12
D5	--	D4	2.1	PD5	D5 - GPIO14
D6	--	D3	2.2	PD6	D2 - GPIO4
D7	--	D2	2.3	PD7	D1 - GPIO5
Pot	--	A0	0.0	PC0	A0
Servo	--	D8	1.7	PB1	D0 - GPIO16
Button	2	D9	1.4	--	D7 - GPIO13
Solenoid	LED	D10	1.5	--	SD3 - GPO10

Results

TinyFPGA Bx

Why Chosen

The *TinyFPGA Bx* was chosen to gain experience with FPGAs and Verilog. This particular FPGA was chosen due to size and cost. FPGAs can be expensive so a $43 board was worth a try. The board size was similar to the size of an *Arduino Nano* board. The details for the *TinyFPGA Bx* are shown below in Table 2. The reason an FPGA was chosen for this project is because FPGAs are more flexible than most microcontrollers – the entire device can be reprogrammed to do any logic task that can be fitted into the number of gates that it has (rscasny, 2018).

Table 2: *TinyFPGA* Details

Compatible Operating Systems	Linux, Windows, MacOS - Used Linux
Cost	$76.42 = $42.99 Board + $15 Breakout board + $11.77 SPI LCD Backpack + $6.66 SPI ADC
Size	18mm x 36mm (66.04 mm x 25.4 mm with breakout board)
Cable	USB Nano B Male
IDE	Atom/Icestudio
Programming Language	Verilog
Features	41 user IO pins 8 MBit of SPI Flash 3.3 V (300 mA) and 1.2 V (150 mA) LDO regulators

Project Progress

The *TinyFPGA* only successfully demonstrated digital in/out using a button and LED.

Complications

- The *TinyFPGA* will inconsistently give a timeout error upon upload attempt. Atom could not be used to upload programs due to this error. The terminal and Icestudio would intermittently work to program the *TinyFPGA*. A comment was left on a GitHub post from September 2018 of someone having the same error but no one responded. The post can be found here: https://github.com/tinyfpga/TinyFPGA-Bootloader/issues/28
- An input such as a button must have an external pull down resistor; without a pull-down resistor, unexpected external factors such as tapping a foot on the ground might cause the output to occur - the LED to turn on.
- The *TinyFPGA* has steep learning curves for learning Verilog, FPGA, SPI, and Icstudio.

Conclusions

The *TinyFPGA* is a very cheap way to learn about FPGAs, even with the added cost of external modules. Projects using the *TinyFPGA* should be limited to digital input and output. With just digital signals, no external modules would be needed and using Icestudio would be straightforward. Using the SPI on this board required a breakout board to easily access the pins, a SPI LCD backpack, and a SPI ADC. There were also no easy examples to follow with using the *TinyFPGA* as the master device for SPI communication. A testbench is used with an FPGA to see the output signal based on the testbench input values. Without a testbench, debugging this FPGA was very difficult. The Icestudio visual programming aspect abstracted the verilog programming which made simple programs easier to accomplish for a beginner.

PSoC 5LP

<u>Why Chosen</u>

The programmable system-on-chip (*PSoC*) was chosen for this project because of previous experience using this development board and IDE in a robotics course. The engineering department at ASU Poly had elected to use the *PSoC* as the development board for many of its classes, and I was interested in determining why this was the case. The details for the *PSoC 5LP* are shown below in Table 3.

Table 3: *PSoC 5LP* Details

Compatible Operating Systems	Windows
Cost	$12.86
Size	315 L x 35 H x 315 W (Mils)
Cable	USB Standard A Female
IDE	*PSoC* Creator Integrated Design Environment (IDE)
Programming Language	C/Visual modules
Features	Max. Operating Voltage: 5.50 V Digital Pins: 38 Analog Blocks: 4 I2C, OpAmps, Timer/Counter/PWM, Comparators

<u>Project Progress</u>

The *PSoC 5LP* was able to complete the whole project. The button, solenoid, potentiometer, servo, and LCD screen are all functioning with the *PSoC 5LP*. Serial communication was also implemented for additional debugging assistance.

<u>Complications</u>

- Booting into Windows to use the PSoC IDE resulted in corrupt boot files. This was not caused by the *PSoC* but made working with this platform difficult.

<u>Conclusions</u>

The PSoC Creator IDE is very complicated and a challenge to initially learn. However, no complications emerged from this learning curve because of previous experience with the platform. The combinations of visual/block modules and C programming is initially difficult to grasp so the PSoC 5LP would not be a beginner development board. The *PSoC 5LP* did successfully complete this project. So for hobbyists looking to learn a new platform, this system will accomplish these tasks. Project difficulties did arise due to using a different operating system. If you do not want to use Windows or have problems doing such, do not use the PSoC 5LP as PSoC Creator is only available on Windows.

Arduino Nano

<u>Why Chosen</u>

The *Arduino Nano* was chosen because the Arduino platform is well-known and very popular with hobbyists. The Nano model was specifically chosen because a couple of Nano boards had been purchased for a previous project and could be repurposed. Furthermore, I had previously used the Arduino Uno and was curious if the smaller board could complete the same functions. The details for the *Arduino Nano* are shown below in Table 4.

Table 4: *Arduino Nano* Details

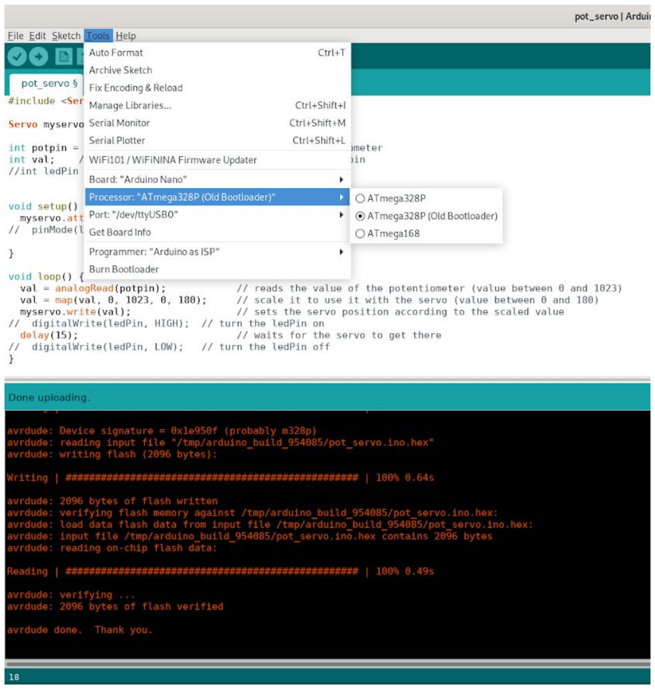
Compatible Operating Systems	Linux, Windows, MacOS - Used Linux
Cost	$22
Size	18 x 45 mm
Cable	USB Mini B Male
IDE	Arduino IDE
Programming Language	C/C++
Features	Operating Voltage: 5V Input Voltage: 7-12 V Digital I/O Pins: 22 Analog IN Pins: 8 PWM Output: 6

<u>Project Progress</u>

The *Arduino Nano* was able to complete the whole project. The button, solenoid, potentiometer, servo, and LCD screen are all functioning with the *Arduino Nano*.

<u>Complications</u>

- Device would not program upon initial use. Selecting “Old Bootloader” option under the Processor menu resolved this issue. See Figure 2.


pot_servo | Ardui
File Edit Sketch Tools Help
pot_servo §
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Get Board Info
Burn Bootloader
ATmega328P
ATmega328P (Old Bootloader)
ATmega168
Done uploading.
18

Board: "Arduino Nano"
Processor: "ATmega328P (Old Bootloader)"
Port: "/dev/ttyUSB0"
Programmer: "Arduino as ISP"
Processor: ○ ATmega328P [✓ ● ATmega328P (Old Bootloader)] ○ ATmega168

```
#include <Ser

Servo myservo

int potpin =                                                  ometer
int val;    /                                                 pin
//int ledPin


void setup()
  myservo.att
//  pinMode(l

}

void loop() {
  val = analogRead(potpin);          // reads the value of the potentiometer (value between 0 and 1023)
  val = map(val, 0, 1023, 0, 180);     // scale it to use it with the servo (value between 0 and 180)
  myservo.write(val);                  // sets the servo position according to the scaled value
//  digitalWrite(ledPin, HIGH);  // turn the ledPin on
  delay(15);                           // waits for the servo to get there
//  digitalWrite(ledPin, LOW);   // turn the ledPin off
}
```

```
avrdude: Device signature = 0x1e950f (probably m328p)
avrdude: reading input file "/tmp/arduino_build_954085/pot_servo.ino.hex"
avrdude: writing flash (2096 bytes):

Writing | ################################################## | 100% 0.64s

avrdude: 2096 bytes of flash written
avrdude: verifying flash memory against /tmp/arduino_build_954085/pot_servo.ino.hex:
avrdude: load data flash data from input file /tmp/arduino_build_954085/pot_servo.ino.hex:
avrdude: input file /tmp/arduino_build_954085/pot_servo.ino.hex contains 2096 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 0.49s

avrdude: verifying ...
avrdude: 2096 bytes of flash verified

avrdude done.  Thank you.
```

Figure 2: Set Processor to ‘Old Bootloader’

Conclusions

The *Arduino Nano* is an inexpensive option that provides enough features to not require any additional hardware to complete a simple robotics project. The IDE is relatively easy to use with only a small setback with having to set the processor. The Arduino SDK is easy to read and has extensive documentation and community support. Numerous examples were available online for each aspect of the project.

Trynkit Husky

Why Chosen

The *Trynkit Husky* was chosen for this project because I am a part of the hardware development team for the startup company. The company was just recently started by ASU Polytechnic students and the board and IDE are still in the development phase. This project was a good way to get fully emerged in the board and help find bugs on the platform. The details for the *Trynkit Husky* are shown below in Table 5.

Table 5: *Trynkit Husky* Details

Compatible Operating Systems	Linux, Windows, MacOS - Used Linux
Cost	$30
Size	82 mm x 35 mm
Cable	None: Programming - Bluetooth/WiFi | Power - LiPo Battery
IDE	Trynkit Web-based IDE
Programming Language	C (future Arduino libraries)
Features	Operating Voltage: 3.5 - 5V Input Voltage: 3.5 - 5V Digital I/O Pins: 14 (6 digital/analog) Analog IN Pins: 6 PWM Output: 6 USART, I2C, SPI, WiFi, BLE, TLS, HTTPS, HTTP, Serial Bluetooth

Project Progress

The full project currently does not work on the *Trynkit Husky*. Digital in/out works with a button and LED. Analog in/out works with a potentiometer and servo, although is very slow. The LCD screen functions to print string values.

Complications

- Currently lacks support for the common C utilities such as sprintf.
- Currently lacks support for Arduino standard libraries.
- Code functionality is inconsistent due to continuing hardware and IDE changes.
- Time limitations restricted the ability to obtain the new hardware with Arduino library capabilities.

Conclusions

The *Trynkit Husky* is an inexpensive option, especially considering its WiFi and BLE capabilities. Setting up the IDE was easy, as it just required logging into a website. The IDE currently does not have many functionalities that make programming easier such as shortcuts but these should be implemented in the future. Currently, the *Husky* only supports programming in low-level, register-based C. This was very difficult as it required intensive referencing of the datasheet to which undescriptively-named registers need to be set to perform each operation. Programming the *Husky* required learning about interrupt service routines (ISRs) and hardware timers to perform simple tasks such as establishing a PWM signal. This board, in its current state, would be a challenge for users without prior C and microcontroller knowledge. The *Trynkit Husky* currently has limited debugging capabilities so other platforms may be easier for those looking to learn C.

In conclusion, the *Trynkit Husky* seems like a great, emerging development board but it still needs more time to become reliable and to contain all the functionality of more established development boards. In the future, this board will be much easier to program using the Arduino libraries. Then, the ability to program using BLE or WiFi would be beneficial, especially for projects that require enclosing the board.

NodeMCU

Why Chosen

The *NodeMCU* is a breakout board for the ESP8266 target chip. This board was chosen for this project after the complications with the *TinyFPGA* programming arose. A colleague had used this board for his honors thesis and recommended it for this project. Because the board is programmed through command line utilities, any IDE or text editor can be used for development with the correct configuration. The IDE I used was a JetBrains product, CLion. I already had experience with another JetBrains product, PyCharm, using it for Python and I was interested to see how CLion compared. The details for the *NodeMCU* are shown below in Table 6.

Table 6: *NodeMCU* Details

Compatible Operating Systems	Linux, Windows, MacOS - Used Linux
Cost	$8.39
Size	49 x 24.5 x 13 mm
Cable	USB Micro B Male
IDE	Any text editor − Used CLion/Platformio
Programming Language	Arduino SDK/C/C++
Features	Power: 5V Digital I/O: 13 Analog Input: 1 WiFi, SPI, UART

Project Progress

The *NodeMCU* was able to complete the whole project. The button, solenoid, potentiometer, servo, and LCD screen are all functioning with the *NodeMCU*.

Complications

- The *NodeMCU* can be programmed using Arduino libraries with Platformio. Platformio is used to manage dependencies, libraries, toolkits, and debuggers in a common way across different board architectures. *#include <Arduino.h>* needs to be included at the top of the program after installation of the Arduino-ESP8266 libraries, whereas this is abstractly included when using the Arduino IDE.
- The *NodeMCU* pinout does not match the pinout for the ESP8266 target chip on board. To control an LED plugged into D8, the software would have to program GPIO15. Pinout shown in Figure 3.
- GPIO 6, 7, 8, and 11 are not available on the *NodeMCU*.
- GPIO 0, 2, and 15 are programming pins and cannot be connected during programming. They can be used as GPIO pins if connected post-upload.

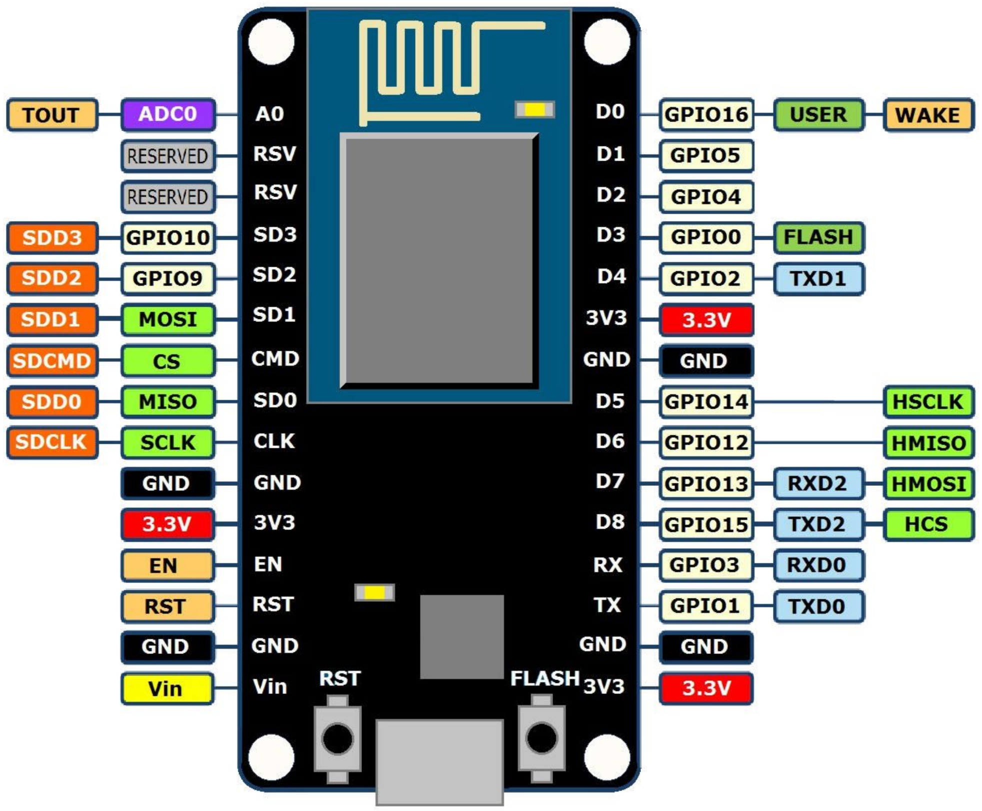


Figure 3: NodeMCU Pinout

<u>Conclusions</u>

Minus a few frustrations, this board successfully completed the project. The CLion IDE was easy to use because of prior use with JetBrains IDEs. The IDE usage was pretty straight forward and had documentation so someone without prior experience would be able to use the platform. The use of Arduino libraries made the programming easy as soon as the correct pinout was found. Users should be aware that if their project requires a lot of GPIO pins, this board may not fit their needs, having only 10 that are always available. However, for under $9 this board is very affordable. This board also contains WiFi capabilities, and although this was not used for this project, this chip is a very inexpensive board to contain that feature.

Comparison

Arduino Nano was the easiest development board to use for this project. The IDE was easy to use and worked consistently. The programming language was easy to understand and plenty of reference examples existed. While the *Arduino Nano* was the 3rd most expensive board, is was the easiest to implement and resulted in a working product the quickest.

With the use of Platformio, the *NodeMCU* acquired the ease of Arduino libraries for programming. This platform also allowed for a choice in IDE. The chosen CLion IDE was easy to use and provides a reference for using other JetBrains IDEs such as PyCharm. While *NodeMCU* is cheaper than the *Arduino Nano*, it has fewer IO pins and the pin labeling is not intuitive. The *NodeMCU* would be superior to the *Arduino Nano* for projects using WiFi as the *Nano* does not have WiFi capabilities.

The *PSoC 5LP* was limited to use with Windows. As a Linux user this made its use prohibitive and frustrating. The IDE and provided libraries are also not as intuitive as using Arduino libraries. Unless the user's desired operating system is not a Windows OS, the PSoC 5LP is a good, inexpensive board with a variety of features.

The Trynkit Husky is still in the early phases of development. The IDE does not have some of the nice features of a more established IDE. The only available programming language at the time of this project was C without the capability to use Arduino libraries. The IDE and programming capabilities were inconsistent due to the board and IDE being under development. Code that worked one week, may not work the next. Using a battery was not ideal when developing code because development would halt when the battery lost charge. This development board would be useful once the set up is more established because the board can be programmed through BLE or WiFi instead of a wired connection. This allows the board to stay in an enclosure when programmed rather than having to dismantle a project just to reprogram the board. The *NodeMCU* is the only other board with the possibility of this capability because of the WiFi capabilities of its ESP8266.

The *TinyFPGA* was inferior compared to the other development boards for this project. The *TinyFPGA* was the most expensive and was the most difficult to use. The programming language and IDE had significant learning curves. Only having digital IO pins required the use of SPI for any analog communication. These learning curves and the inconsistency of the system ultimately caused the project to not be completed on this board by the deadline.

Conclusion

In conclusion, the *Arduino Nano* was the easiest platform to use. The *NodeMCU* was a very affordable option with limited IO pins. Unless the user's desired operating system is not a Windows OS, the PSoC 5LP is a good, inexpensive board that requires additional time to learn a more complicated IDE. The *Trynkit Husky* had limitations due to being in its development phase; however, it could be beneficial in the future for those wishing to program their development board wirelessly. The *TinyFPGA* was inferior to the other development boards for the project outlined in this paper due to cost and ease of use. The *TinyFPGA* was a cheaper way to learn about FPGAs and would be useful in a project of only digital inputs and outputs.

Limitations/Future Works

The most hindering limitation for this project was available time. Future work for this project would include learning how to use SPI on the *TinyFPGA*. The *TinyFPGA* could be used in a project as the slave in SPI communication with a microcontroller. The *TinyFPGA* could also be used for a project using solely digital inputs and outputs such as a lighting project. FPGAs are more focused on digital controls so a modified project could do a better job of assessing its intended functionality. The *Trynkit Husky* should be revisited when the IDE and board are more established to better assess how it compares against the already established companies. Continuing this comparison could include adding a rack and pinion to the robot to add vertical control.

Note

For more information, go to the GitHub repository for this project:
https://github.com/KyKyPi/embedded-platform-comparison

References

Nodemcu pinout. (2018, July 22). Retrieved March 3, 2019, from https://circuits4you.com/2017/12/31/nodemcu-pinout/

rscasny. (2018, February 22). Comparing an FPGA to a Microcontroller, Microprocessor or an ASIC. Retrieved January 10, 2019, from https://www.element14.com/community/groups/fpga-group/blog/2018/02/22/comparing-an-fpga-to-a-microcontroller-microprocessor-or-an-asic

Teel, J. (2018, May 28). How to Choose the Best Development Kit: The Ultimate Guide for Beginners. Retrieved January 11, 2019, from https://predictabledesigns.com/how-to-choose-the-best-development-kit-the-ultimate-guide-for-beginners/